Automatic Flow Chart Design

G. Hain
Dynaelectron Corporation

Klaus Hain
Theoretical Division
Goddard Space Flight Center
National Aeronautics and Space Administration
Greenbelt, Maryland

ABSTRACT

18335

There is an increasing demand for good documentation of complex programs; in particular, the design of flow charts takes a lot of time and is a strain on the programmer. For these reasons, flow charts are often neglected. This paper describes a program written in Fortran that does the tiresome work of drawing the flow chart of a program. The actual drawing is done with the aid of a SC 4020 and can easily be changed for any other plotter device. Such a flow chart gives the logical structure of a program, e.g., it shows the connections between the different parts of a program and its branches. This program not only can be used in the documentation of programs, but also to find logical errors and also to "clean up" (reorganize) a program which is in the stages of development. The same program can be used to draw PERT networks.

Author

[redacted]

INTRODUCTION

A program can be logically described, as a set of elements A_j, which contain the instructions, and a matrix C_{jk} which gives the connections between these elements (JUMP instructions). (In PERT terms the elements A_j are events, and the $C_{j,i}$ activities.) Here the attempt is made to draw a two-dimensional picture of the logical structure of a program by disregarding the specific contents of the elements. To get this picture two facts are important.

a) There is a hierarchy, namely the order by which the elements are called. This order would be unique if there were no closed loops. Therefore, the first aim is to achieve this unique order by handling the loops appropriately and then setting up the sequential order. (Level structure, y-coordinates.)

b) Usually there will be more than one element is one of these levels, so that the task is now to sort the elements in such a way that there will be as little crossing of connecting lines as possible; furthermore, similar elements shall be brought as close together as possible. (Branch structure, x-coordinates).

1) LEVEL STRUCTURE

Given a set of elements $A_1 \ldots A_n$ and a connecting matrix, $C_{j,1} = j_1, \ldots, C_{j,m} = j_m$ if element A_j calls the elements $A_{j1} \ldots A_{jm}$.

There may exist some elements for which $m = 0$, which means that those elements do not call any other element. These elements are defined as end elements.

When the call matrix is given one can construct uniquely the reciprocal "get" - matrix $G_{j,k}$ - which is defined by $G_{j,k} = j_1, \cdot\cdot, G_{j,m} = j_m$ in which the element A_j is called by the elements $A_{j1} \ldots A_{j_m}$. An element for which $m = 0$ is called an entry element.

DEFINITION:

A <u>program</u> is defined as a set of elements $A_1 \ldots A_n$ which are blocks of instructions with the connecting matrix $C_{j,k}$ resp. $G_{j,k}$ if the following conditions are fulfilled.

1) there exists one and only one entry element.
2) there exists at least one end element.
3) there exists for each element j a number $m \leq n$, such that $(G_{j,k})^m$ contains the entry element (from every element there must exist at least one path to the entry point).
4) there exists for each element j a number $m \leq n$, such that $(C_{j,k})^m$ contains at least one end element (from every element there must exist at least one path to an end element).

In the following, the elements $A_1 \ldots A_n$ are represented by their subscripts. In order to be able to achieve a unique hierarchy of the elements in a simple way it is useful to break up the loops so that the calling sequence runs only in one direction.

DEFINITION:

If $(C_{j,k})^m$ has m diagonal elements and m is the first number for which diagonal elements occur, then the diagonal elements form a simple loop of mth order. (One can prove that if one element $(C_{j,k})^m$ is a diagonal element, then the whole set of elements which are involved have the same property. Also that the definition with the matrix $(C_{j,k})^m$ is equivalent to this definition.) If the number of diagonal elements is bigger than m, then they form a multiple loop of mth order.

DEFINITION:

A program is called cyclic, if it contains at least one loop of order m, $2 \leq m \leq n - 2$, and noncyclic if it does not contain any such loops.

In order to handle a program as a sequential structure it has to be noncyclic. Therefore a cyclic program has to be made noncyclic by breaking up the loops. This can be done by introducing a new end element in the following way.

Lemma: There exists at least one element of a loop of mth order which is called by at least one element which is not an element of this loop and also at least one element which calls an element, which is not part of the loop.

Proof: If $(C_{j,k})^m$ has only loop elements, then any potent of $(C_{j,k})^m$ would only have loop elements, which would contradict (4). If $(C_{j,k})^m$ is to have at least one element which is not a loop element, at least one $C_{j,k}$ of a loop element has to have one element which is no part of the loop. As $(C_{j,k})^m$ and $(G_{j,k})^m$ are equivalent in defining a loop the same arguments apply to the matrix $G_{j,k}$.

Let A_j be an element, which is called by an element which is not a loop element. Then a new element A_{n+1} is introduced (with the name $-A_j$) and a new $C_{n+1,k} = 0$ is obtained as a new end element A_{n+1}. $G_{j,k}$ in which k is a member of the loop is then replaced by $G_{n+1,k}$.

If possible the same end element is used for different simple loops which are parts of a multiple loop and also for loops of different order.

After all loops are broken up the program is no longer cyclic. For noncyclic programs, it is possible to define a unique sequential structure called the level structure.

DEFINITION:

The level ℓ_j of an element is defined by:

1) The entry element has the level 1.

2) If an element k, which is called by j and other elements $j_1 \ldots j_\sigma$ with levels $\ell_{j1}, \ldots, \ell_{j_\sigma} \leq \ell_j$, then $\ell_k = \ell_j + 1$.

Lemma: The levels are uniquely defined.

Proof: If one element j were to have two different levels ℓ_j and $\overline{\ell}_j$ then there must exist also at least one element k which calls the element j whose level is not uniquely defined. So that at least the entry element can be reached and therefore the level of the entry element would not be uniquely defined which contradicts 1).

In the drawing of the flow charts each element is assigned a y-coordinate according to its level

$$y_j = A \cdot \ell_j$$

2) BRANCH STRUCTURE

After one has accomplished the hierarchial structure one has to sort the elements of the same level. The sorting should give a clear picture of branch structure. Therefore the aim is to avoid the crossing of lines as much as possible, which is equivalent to minimizing the distance between the elements.

The elements 1, ... n, with the $y_1 \ldots y_n$ and the connecting lines $C_{j,k}$ are given. These can also be represented by $S_{j,k} = C_{j,k} + G_{j,k}$ $(S_{j,k})$ is the matrix which contains all possible connections between the elements . Given furthermore a weight function $H_{j,k}(y_j - y_k)$, then the task could be principally defined as

$$\sum_{j,k} S_{j,k} \, (x_j - x_k)^2 \, H_{j,k}(y_j - y_k) = \text{Minimum}$$

by varying the order in which the elements occur in each level and x_j subject to the restriction that $|x_j - x_k| \geq 1$ if $y_j = y_k$, where $H_{j,k}$ is a weight function which gives more weight to the elements with the least level difference. Furthermore, elements of the same structure should be put together as close as possible. This task can be solved approximately in two steps:

a) Order the elements in a level.

b) Calculate the x-coordinates (in some crude way of linear programming).

Definitions: Ordering - a)

1. identical elements; an element j_1 is identical with an element j_2, if $S_{j1,k} = S_{j2,k}$ $k = 1,\ldots n_{j1}$

2. chain elements. An element j is a chain element if $C_{j,k}$ and $G_{j,k}$ contain only one element.

3. $S^R{}_{j,k}$ is the reduced $S_{j,k}$ and is obtained from $S_{j,k}$ by
   a) only one of the identical elements j is retained and has the multiplicity m_j
   b) chain elements are eliminated from $S_{j,k}$ and identical elements are taken out again.

4. The basic matrix $B^{(1)}_{j,k}$ is the matrix obtained from $S^R{}_{j,k}$ in which the chain elements are restored, but only one element for each level in a specific chain so that an appropriate place for them can be found. To these chain elements the multiplicity m has to be assigned, which corresponds to their appearance in the chain as identical elements.

The sorting is done first for $B^{(2)}_{j,k}$, which is derived from the two time reduced matrix $S^{RR}_{j,k}$. It gives a uniquely defined order for elements of $B_{j,k}$. This order is retained for sorting $S_{j,k}$.

The sorting is done by following a path through the program. It is useful to define the following numbers:

1) a number Z_j which is +1 if j is reached from a lower level and -1, if j is reached from a higher level.

2) K_j is a vector which contains those elements whose order is already determined.

3) $R^a{}_j$ and $R^b{}_j$ are vectors which give the order of an element j. Initially $R^a{}_j = n^a{}_j + 1$, where $n^a{}_j$ is the beginning of level ℓ_j

$$n^a{}_{\ell_j} = \sum_k m_k \text{ with } \ell_k < \ell_j$$

and $R^e{}_j = n^e{}_j$ with

$$n^e{}_{\ell_j} = \sum_k m_k \text{ with } \ell_k \leq \ell_j$$

If $R^a_j - R^e_j = m_j$ the place of an element is determined.

4) d_j is a vector, which contains the elements $S_{j,k}$ from K_j whose level difference is one.

5) m_j is the vector which gives the multiplicity.

Sorting starts at an arbitrary element which is not an only element in a level. The most convenient way is to take an element of the level, which contains the most items, and $Z_{j1} = 1$, $R^e_{j1} = R^q_{j1} + m_{j1}$ and $R^a_j = R^e_{j1}$ if $y_j = y_{j1}$. If the sorting procedure has reached the jth element K_j, the element K_{j+1} is found by the following process

(1) $$K_{j+1} = S_{K_{j\sigma},a} \qquad j_\sigma \leq j$$

if there is an element K_{j_σ} so that K_{j+1} is reached from the same direction as K_j where K_{j_σ} is the first element counting from the end which fulfills the condition $Z_{K_{j_\sigma}} = Z_{K_j}$. If there is not such an element the first element $K_{j+1} = S_{j_{K_\sigma},a}$ then $Z_{K_{j+1}} = -Z_{K_j}$ is taken.

(2) If there exists a $S_{k_{j\sigma,\alpha}}$ with $\ell_{S_{k_{j_{\sigma,\alpha}}}} = \ell_{k_{j+1}}$ and $R^a_{S_{k_{j,\alpha}}} < R^a_{k_{j+1}}$

then $K_{j+1} = S_{k_{j_{\sigma,\alpha}}}$

(3) Furthermore, if now the so defined K_{j+1} is taken and there is a d_k $k = 1\ldots,n_j$, so that

$$R^a_{d_k} \leq R^a_{K_{j+1}} \quad \text{then}$$

$$K_{j+1} = d_k$$

$Z_{k_{j+1}}$ has then the same value, as the element, from which d_k has been originally determined.

(4) The element K_{j+1} is then eliminated out of the matrix B so that it does not appear again.

(5) $R^e_{k_{j+1}} = R^a_{k_{j+1}} + m_{k_{j+1}}$, $R^a_j = R^e_{k_{j+1}}$ if $y_i = y_{k_{j+1}}$

The procedure is so destined to achieve the two aims:

1. to make as rare as possible (by means of the vector d_j) the crossing of lines which connect elements with only one level difference.
2. to bring so close as possible such elements which belong logically together by keeping the vertical direction of the elimination as long as possible.

The actual program is still more complicated in detail, but this is not essential for the main features described here. These details can be seen from the Fortran program.

After this procedure has been applied to this basic matrix $B^{(2)}_{j,k}$; it is repeated for the matrix $B^{(1)}$ and $S_{j,k}$ with the initial values R^a_j, R^e_j given by the just processed basic matrix $B^{(2)}_{j,k}$, $B^{(1)}$ respectively.

(b) Calculation of the x-coordinates

After the sorting is done and every element has obtained its place the actual calculation of the x-coordinates starts. This is done in a rather crude and simple way. The row in the level $\ell = \ell_{max}$ which has the most elements is taken and a matrix $\overset{*}{C}_{j,k}$ is constructed which gives all connections starting from this row. The elements are ordered in the sequence in levels with $e_1 = \ell_{max}$, $e_1 = \ell_{max} + 1$, $e_3 = \ell_{max} - 1, \ldots, e_N$, where N is the number of levels. Inside the levels are ordered according to R^a_j. The elements are now in an unique order. $\overset{*}{C}_{j,k}$ gives then the connections of elements with higher numbers to those with lower numbers. The distance between the elements in the ℓ_{max} row, whose order is given by R^a_j is set arbitrarily to one.

Assume that the x-coordinates have been determined up to the level e_k, then the x-coordinates in e_{k+1} are computed by the following procedure. If j is an element of this level e_k

1) $X_j = \text{Max} = \left(\frac{1}{N_j} \sum_{k=1}^{N_j} x \cdot \overset{*}{C}_{j,k},\ X_{j-1} + 1 \right)$ (N_j is the number of

elements in the $\overset{*}{C}_{j,k}$.

where the order of the elements are given by R^a_j. If j is the first element then X_{j-1} is set artificially to -6.

2) should there be an end element so that $\overset{*}{C}_{j,k} = 0$, this element is brought in with

$$x_j = x_{j-1} + 1$$

and if this j is the first element in a level

then $x_j = x_{j+1} - 1$

3) the $a = \frac{1}{N_{e_k}} \sum_{j=1}^{N_{e_k}} - \frac{1}{N_j} \sum_{j=1}^{N_j} X_{e_{j,k}}$ $X_j \to X_j - a$

$$X_j \to X_j - a$$

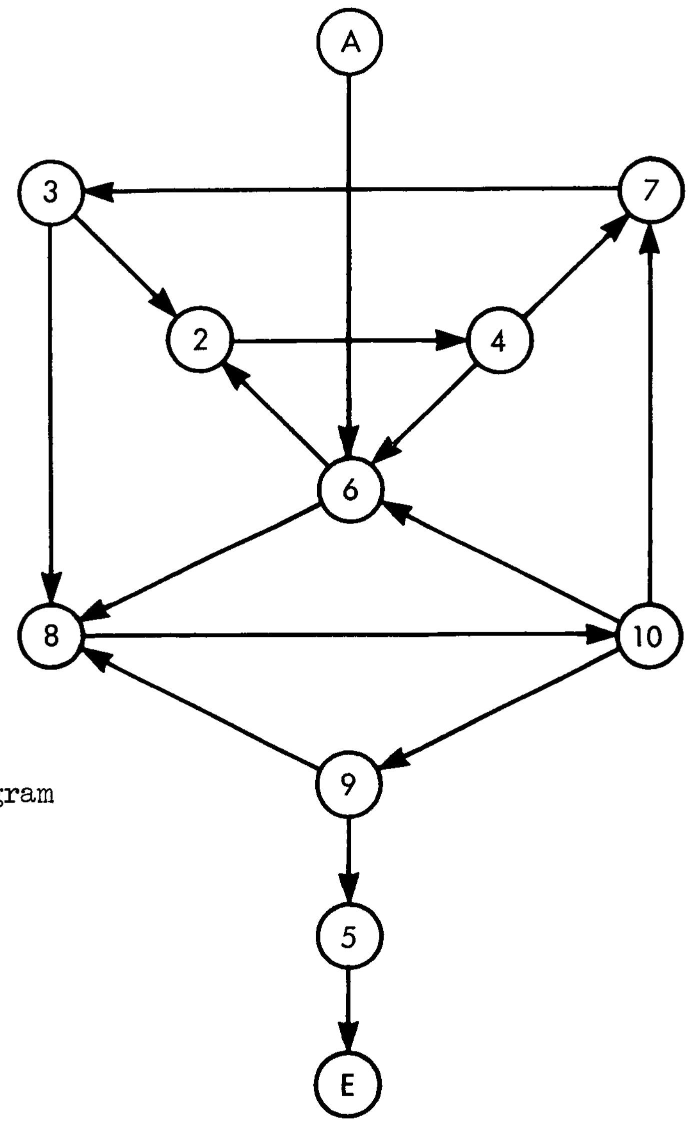
The graphs shown are done by a SC 4020, but the program can be easily changed for use with any other plotter device. If there are many elements the plotting is done on several pages. If there is more than one entry for a program, an artificial entry with name '//' is provided in order to connect the various entries of a program.

The dimension of the program is that two hundred different elements can be handled and one element can be connected to as many as 20 other elements. The program is written in Fortran II and contains approximately 1600 Fortran statements.

In conclusion, it seems that this program (including the graphical output) can be a very powerful tool in documentation, analysis and cleanup of a program. It should also be of use for drawing of big networks like PERT networks.

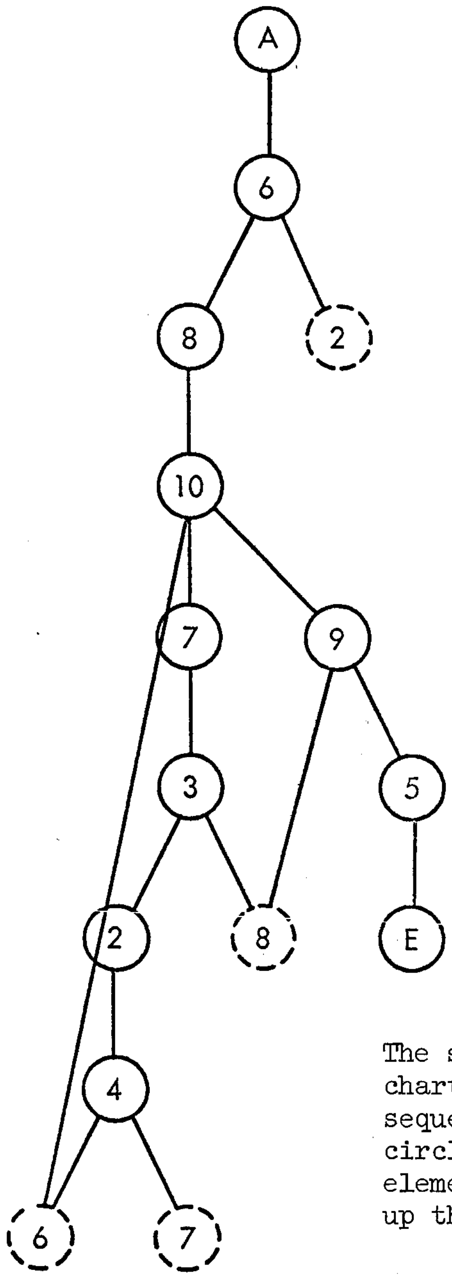
ACKNOWLEDGEMENTS

The authors would like to thank Dr. K. V. Roberts for initiation of the program and for many fruitful discussions. Acknowledgement is also made to the UKAEA for giving the authors the opportunity to spend some weeks at the Culham laboratories.



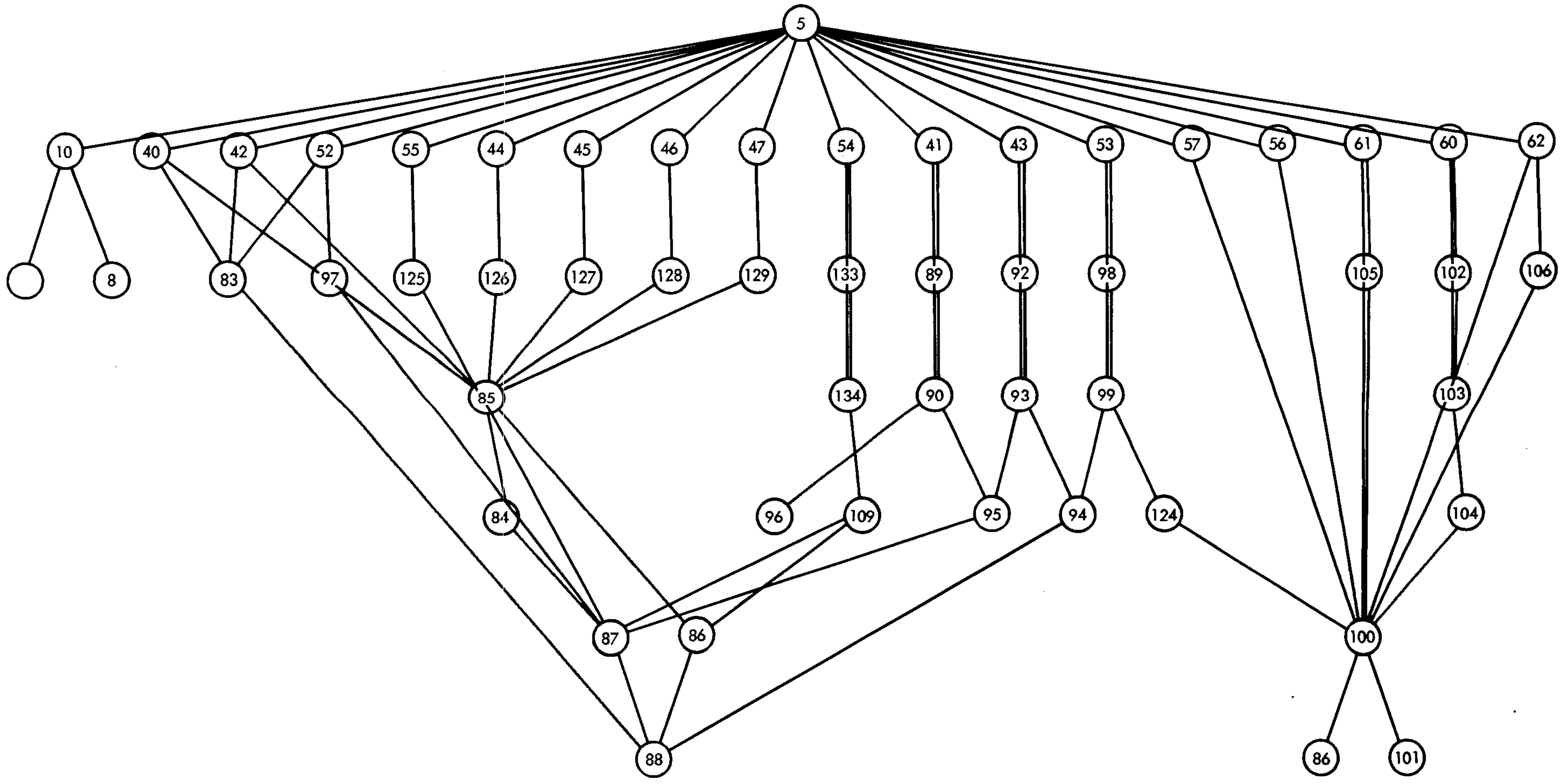
Hand drawn flow shart of a program
with severed multiple loops.

1a



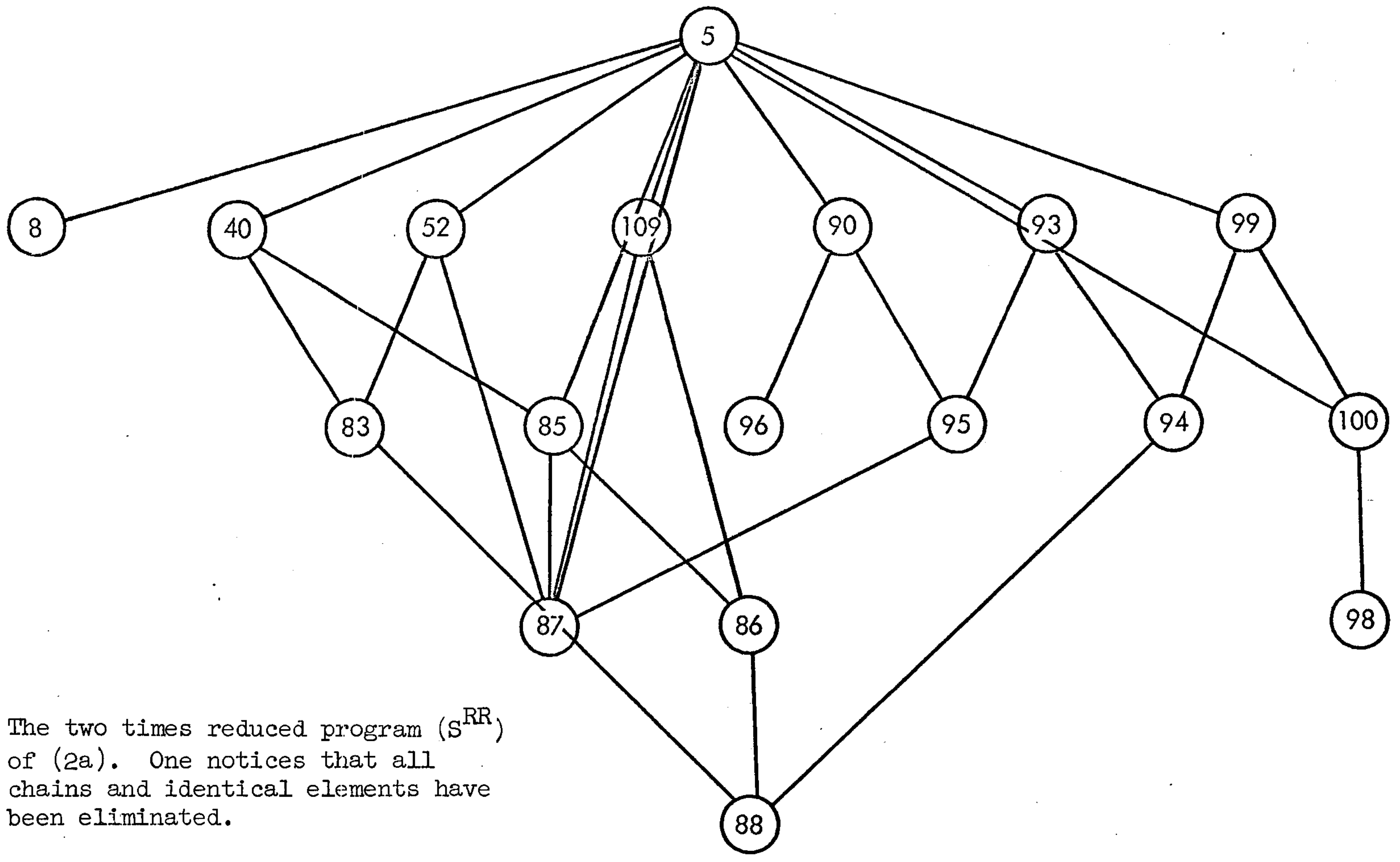
The same program as 1a, the flow chart drawn automatically in a sequential way. The "dotted" circle indicates the newly introduced elements, which are used for breaking up the loops.

1b



Flow Chart of a non-cyclic program.

2a



The two times reduced program (S^{RR}) of (2a). One notices that all chains and identical elements have been eliminated.

2 b